Multi-Target and Structured-Output Learning

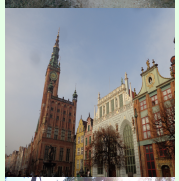
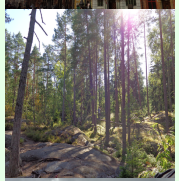
INF581 Advanced Machine Learning and Autonomous Agents

Jesse Read

Last Updated: January 12, 2022

Multi-Label Classification

Multi-Label Classification

Input	Beach	Sunset	Foliage	Urban
	1	0	1	0
	0	1	0	0
	0	1	0	1
	0	1	1	0
	0	0	1	1
	?	?	?	?

Given an instance $\boldsymbol{x}$, we obtain predictions $\widehat{\boldsymbol{y}} = h(\boldsymbol{x})$.

The Lord of the Rings: The Fellowship of the Ring (2001)

Top 500

PG-13 | 178 min | Adventure, Fantasy | 19 December 2001 (USA)

8.8

Your rating: -/10

Ratings: **8.8**/10 from 1,110,948 users Metascore: 92/100

Reviews: 4,988 user | 294 critic | 34 from Metacritic.com

A meek hobbit of the Shire and eight companions set out on a journey to Mount Doom to destroy the One Ring and the dark lord Sauron.

Director: Peter Jackson

Writers: J.R.R. Tolkien (novel), Fran Walsh (screenplay), 2 more credits »

Stars: Elijah Wood, Ian McKellen, Orlando Bloom |

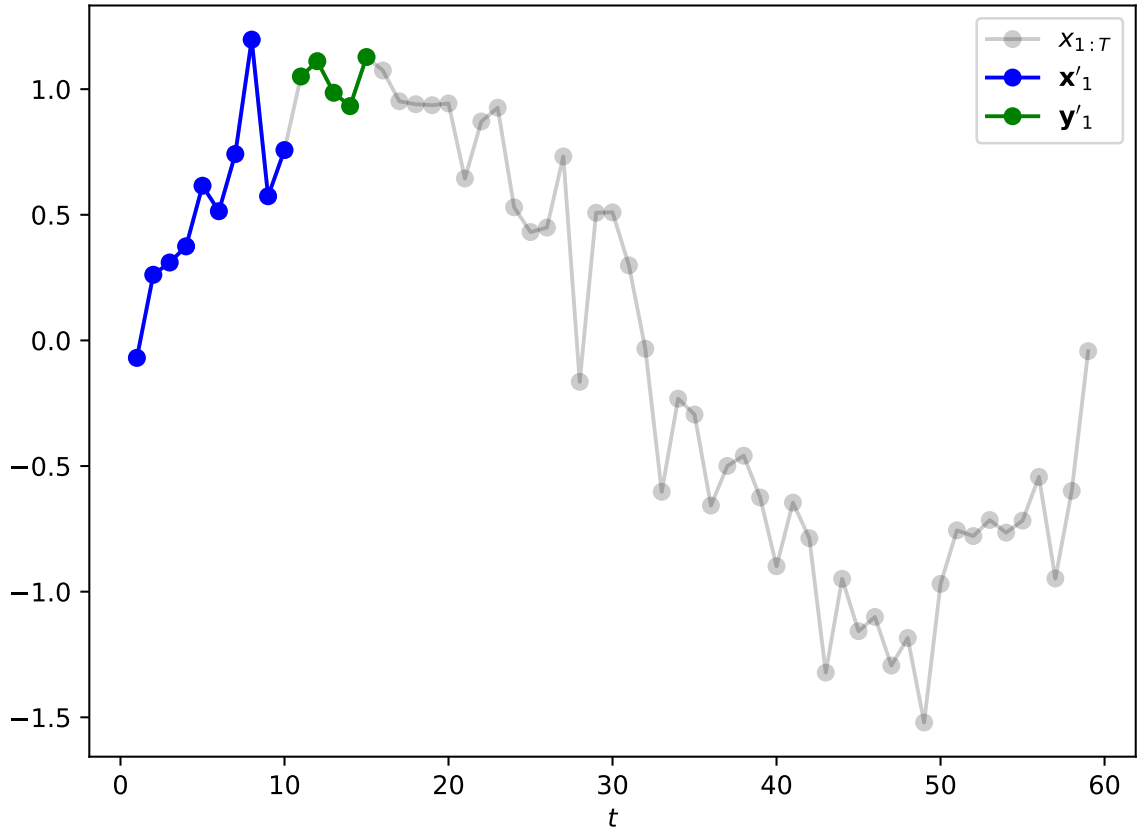
See full cast and crew »

Source: [1]

Missing-data Imputation / Recommender Systems

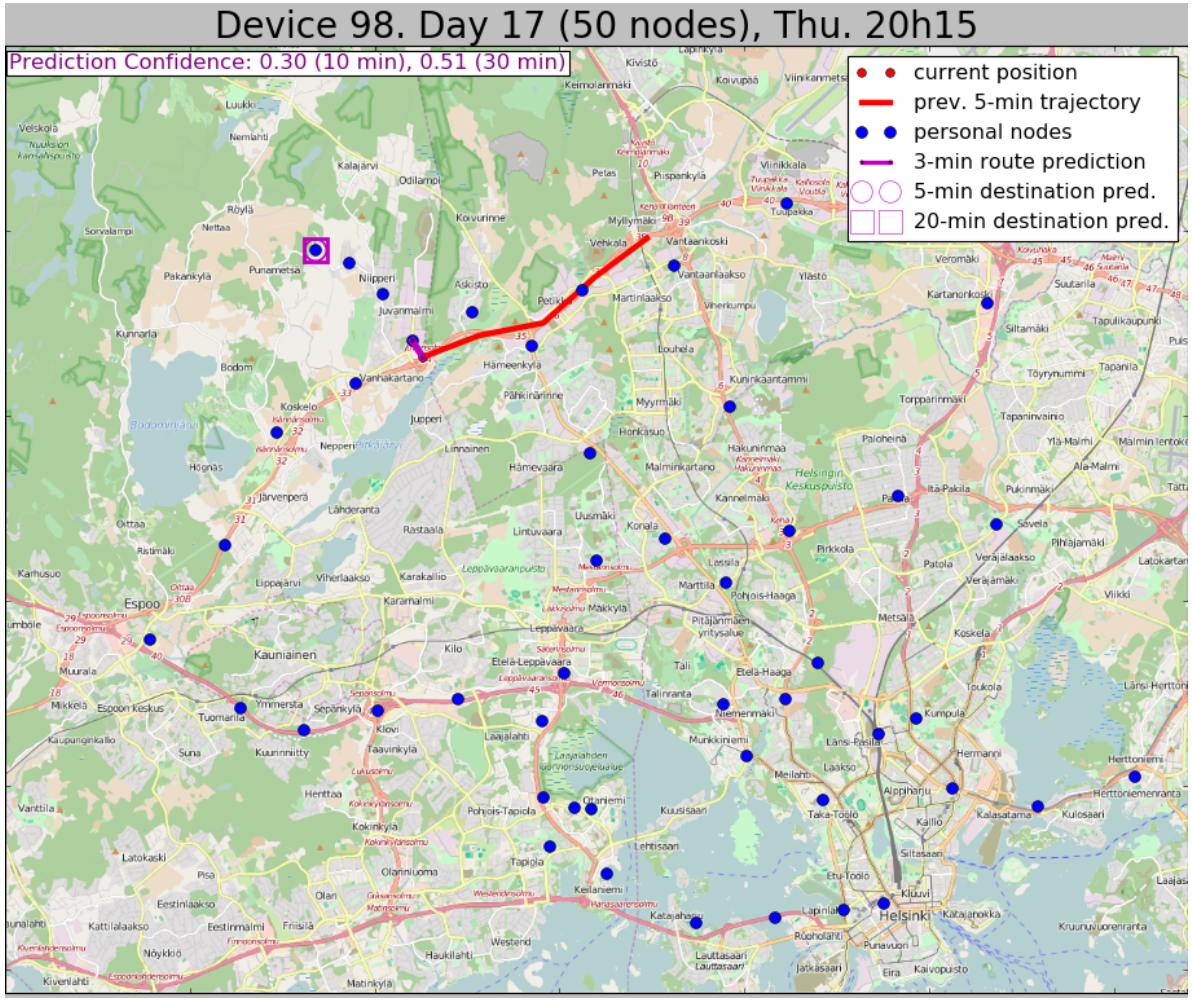
	Film 2 X_2	Film 3 X_4	Book 1 X_1	Book 2 X_3	Song 5 X_5
	0	0	1	1	0
	1	1	?	0	?
	0	0	1	0	0
	1	1	?	0	1
	0	0	0	?	?
	1	0	?	1	?

Time Series Forecasting



(including multi-dimensional time series; generalization to $\boldsymbol{y} \in \mathbb{R}^m$).

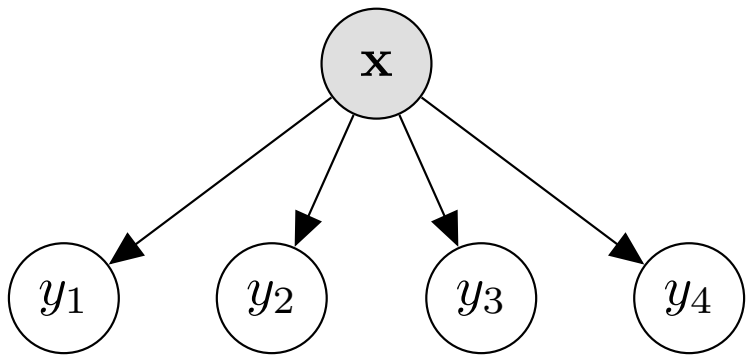
Trajectory Prediction



Trajectory prediction in urban environment using mobile phone data

Binary Relevance (Independent Classifiers): The Baseline

$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	0
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	0
$\boldsymbol{x}^{(4)}$	1	0	0	1
$\boldsymbol{x}^{(5)}$	0	0	0	1
$\tilde{\boldsymbol{x}}$	?	?	?	?



The binary relevance method = *one binary classifier trained for each label*, i.e., independent models.

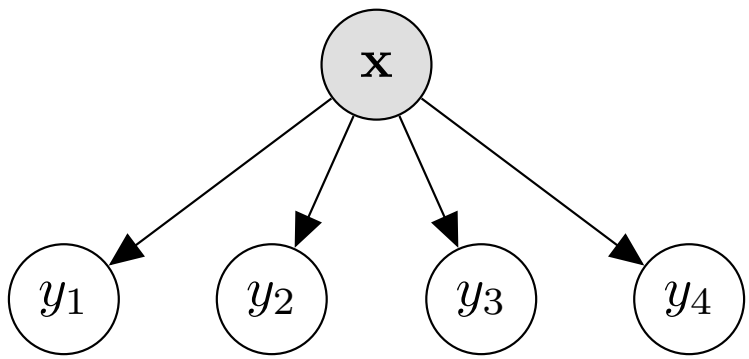
Binary Relevance (Independent Classifiers): The Baseline

$\boldsymbol{X}$	Y_1
$\boldsymbol{x}^{(1)}$	0
$\boldsymbol{x}^{(2)}$	1
$\boldsymbol{x}^{(3)}$	0
$\boldsymbol{x}^{(4)}$	1
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?

$\boldsymbol{X}$	Y_2
$\boldsymbol{x}^{(1)}$	1
$\boldsymbol{x}^{(2)}$	0
$\boldsymbol{x}^{(3)}$	1
$\boldsymbol{x}^{(4)}$	0
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?

$\boldsymbol{X}$	Y_3
$\boldsymbol{x}^{(1)}$	0
$\boldsymbol{x}^{(2)}$	1
$\boldsymbol{x}^{(3)}$	0
$\boldsymbol{x}^{(4)}$	1
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?

$\boldsymbol{X}$	Y_4
$\boldsymbol{x}^{(1)}$	1
$\boldsymbol{x}^{(2)}$	0
$\boldsymbol{x}^{(3)}$	1
$\boldsymbol{x}^{(4)}$	0
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?



The binary relevance method = *one binary classifier trained for each label*, i.e., independent models.

Main Questions (for this Lecture)

When is binary relevance (independent models) *not* the best approach to take?

And in that case, when independent models are not optimal,

- Which structure, how to learn it
- How to performing inference in that structure

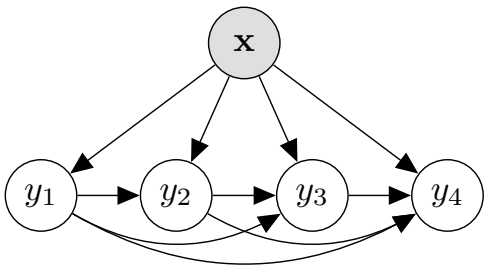
And why does this work better than independent models?

'Bonus question': Overlap between probabilistic and deep neural models

Classifier Chains

Classifier Chains (and Greedy Inference)

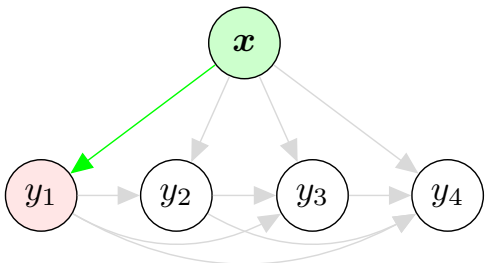
A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. Each $P(Y_j|\text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y_1}$	$\widehat{y_2}$	$\widehat{y_3}$	$\widehat{y_4}$

Classifier Chains (and Greedy Inference)

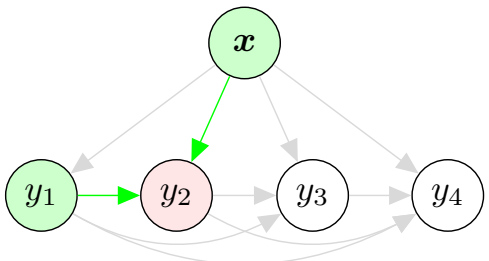
A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. Each $P(Y_j|\text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y_1}$			

Classifier Chains (and Greedy Inference)

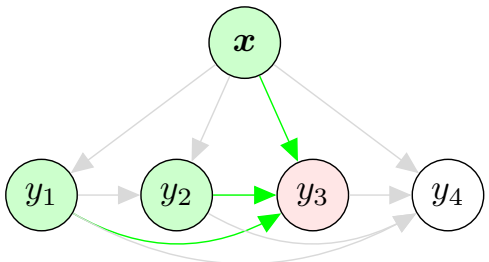
A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. Each $P(Y_j|\text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$		

Classifier Chains (and Greedy Inference)

A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. Each $P(Y_j|\text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, ...). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$	

For example,

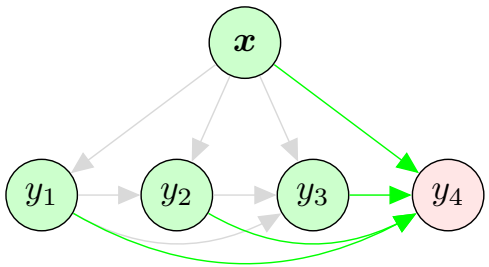
$$\widehat{y}_3 = h_3(\boldsymbol{x}, \widehat{y}_1, \widehat{y}_2) = \underset{y_3 \in \{0,1\}}{\text{argmax}}\, P(y_3|\boldsymbol{x}, \widehat{y}_1, \widehat{y}_2)$$

Use training data to fit base classifier h_3.

Inference: $\widehat{y}_1, \widehat{y}_2, \ldots$ are greedy *predictions* from $h_1, h_2, \ldots$.

Classifier Chains (and Greedy Inference)

A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. Each $P(Y_j|\text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, ...). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$	$\widehat{y}_4$

For example,

$$\widehat{y}_3 = h_3(\boldsymbol{x}, \widehat{y}_1, \widehat{y}_2) = \underset{y_3 \in \{0,1\}}{\text{argmax}}\, P(y_3|\boldsymbol{x}, \widehat{y}_1, \widehat{y}_2)$$

Use training data to fit base classifier h_3.

Inference: $\widehat{y}_1, \widehat{y}_2, \ldots$ are greedy *predictions* from $h_1, h_2, \ldots$.

Recap: At training time, a multi-label dataset is transformed into L standard binary classification problems:

$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	0
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	0
$\boldsymbol{x}^{(4)}$	1	0	0	1
$\boldsymbol{x}^{(5)}$	0	0	0	1
$\tilde{\boldsymbol{x}}$	?	?	?	?

At test time, each base classifier h_j provides $\widehat{y}_j \in \{0, 1\}$:

$$\widehat{y}_1 = h_1(\tilde{\boldsymbol{x}}) \quad \triangleright \text{ for } j = 1$$
$$\widehat{y}_j = h_j(\tilde{\boldsymbol{x}}, \widehat{y}_1, \ldots, \widehat{y}_{j-1}) \quad \triangleright \text{ for } j = 2, \ldots, L$$

Multi-output classification:

$$\widehat{\boldsymbol{y}} = [\widehat{y}_1, \ldots, \widehat{y}_L]$$

Recap: At training time, a multi-label dataset is transformed into L standard binary classification problems:

$\boldsymbol{X}$	Y_1
$\boldsymbol{x}^{(1)}$	0
$\boldsymbol{x}^{(2)}$	1
$\boldsymbol{x}^{(3)}$	0
$\boldsymbol{x}^{(4)}$	1
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$

$\boldsymbol{X}$	Y_1	Y_2
$\boldsymbol{x}^{(1)}$	0	1
$\boldsymbol{x}^{(2)}$	1	0
$\boldsymbol{x}^{(3)}$	0	1
$\boldsymbol{x}^{(4)}$	1	0
$\boldsymbol{x}^{(5)}$	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$

$\boldsymbol{X}$	Y_1	Y_2	Y_3
$\boldsymbol{x}^{(1)}$	0	1	1
$\boldsymbol{x}^{(2)}$	1	0	0
$\boldsymbol{x}^{(3)}$	0	1	0
$\boldsymbol{x}^{(4)}$	1	0	0
$\boldsymbol{x}^{(5)}$	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$

$\boldsymbol{X}$	Y_1	Y_3	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	0
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	0
$\boldsymbol{x}^{(4)}$	1	0	0	1
$\boldsymbol{x}^{(5)}$	0	0	0	1
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$	$\widehat{y}_4$

At test time, each base classifier h_j provides $\widehat{y}_j \in \{0, 1\}$:

$$\widehat{y}_1 = h_1(\tilde{\boldsymbol{x}}) \quad \triangleright \text{ for } j = 1$$
$$\widehat{y}_j = h_j(\tilde{\boldsymbol{x}}, \widehat{y}_1, \ldots, \widehat{y}_{j-1}) \quad \triangleright \text{ for } j = 2, \ldots, L$$

Multi-output classification:

$$\widehat{\boldsymbol{y}} = [\widehat{y}_1, \ldots, \widehat{y}_L]$$

Classifier Chains (Bayes Optimal Inference)

Each base classifier has a probabilistic interpretation:

$$\widehat{y}_j = h_j(\boldsymbol{x}) = \underset{y_j \in \{0,1\}}{\operatorname{argmax}} P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})$$

(e.g., logistic regression).

From Bayesian network:

$$\begin{aligned}
\widehat{\boldsymbol{y}} &= \underset{\boldsymbol{y} \in \{0,1\}^L}{\operatorname{argmax}} P(\boldsymbol{y} | \boldsymbol{x}) \\
&= \underset{\boldsymbol{y} \in \{0,1\}^L}{\operatorname{argmax}} P(y_1 | \boldsymbol{x}) \prod_{j=2}^{L} P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})
\end{aligned}$$

This is not the same as greedy inference.

Classifier Chains (Bayes Optimal Inference)

Each base classifier has a probabilistic interpretation:

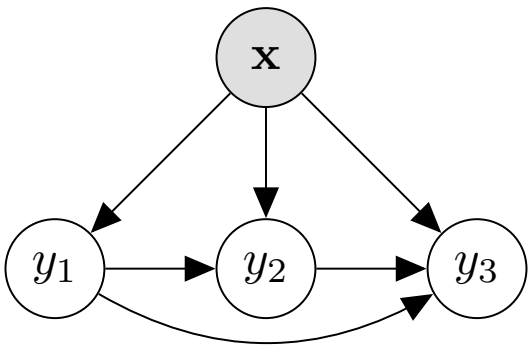
$$\widehat{y}_j = h_j(\boldsymbol{x}) = \underset{y_j \in \{0,1\}}{\operatorname{argmax}} P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})$$

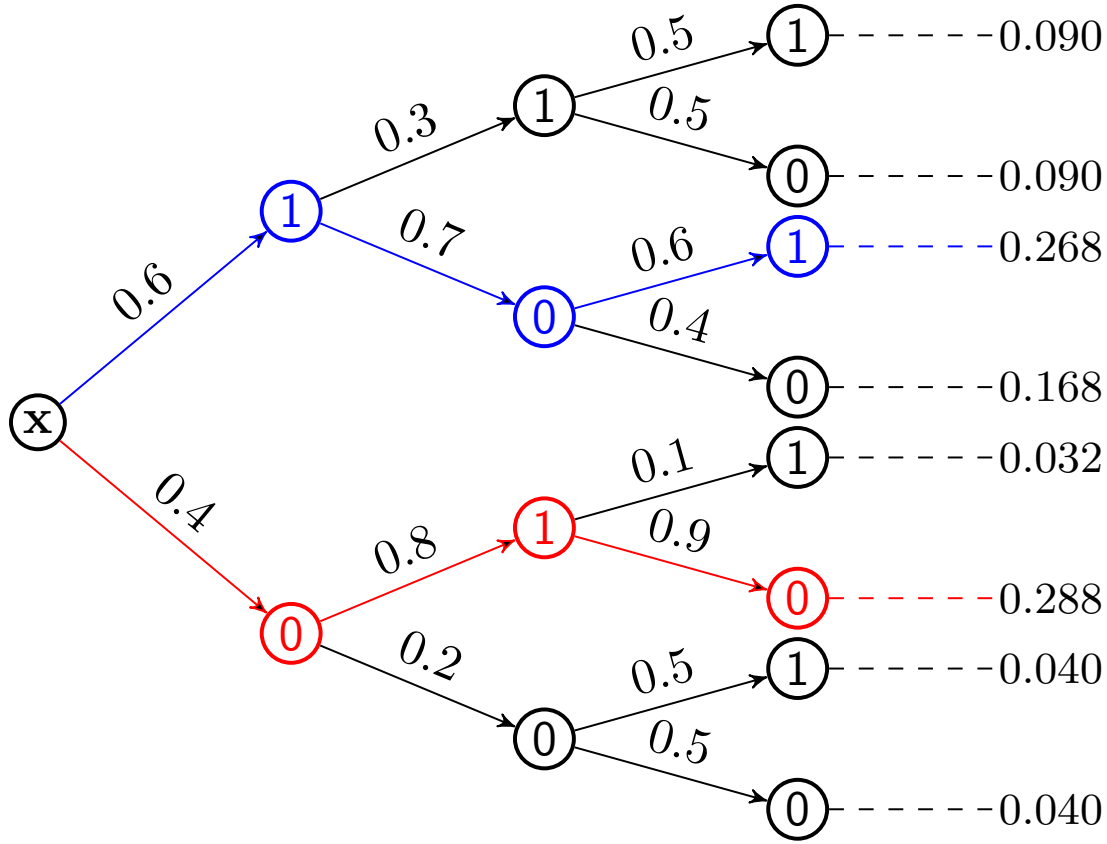
(e.g., logistic regression).

From Bayesian network:

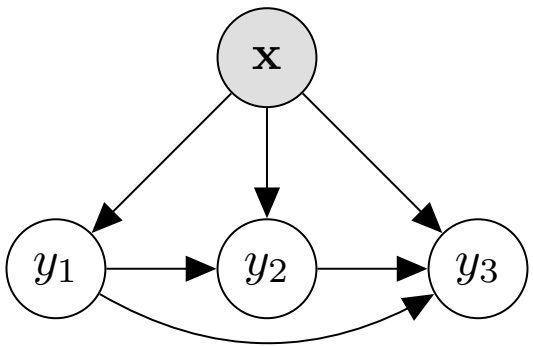
$$\begin{aligned}\widehat{\boldsymbol{y}} &= \underset{\boldsymbol{y} \in \{0,1\}^L}{\operatorname{argmax}} P(\boldsymbol{y} | \boldsymbol{x}) \\ &= \underset{\boldsymbol{y} \in \{0,1\}^L}{\operatorname{argmax}} P(y_1 | \boldsymbol{x}) \prod_{j=2}^{L} P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})\end{aligned}$$

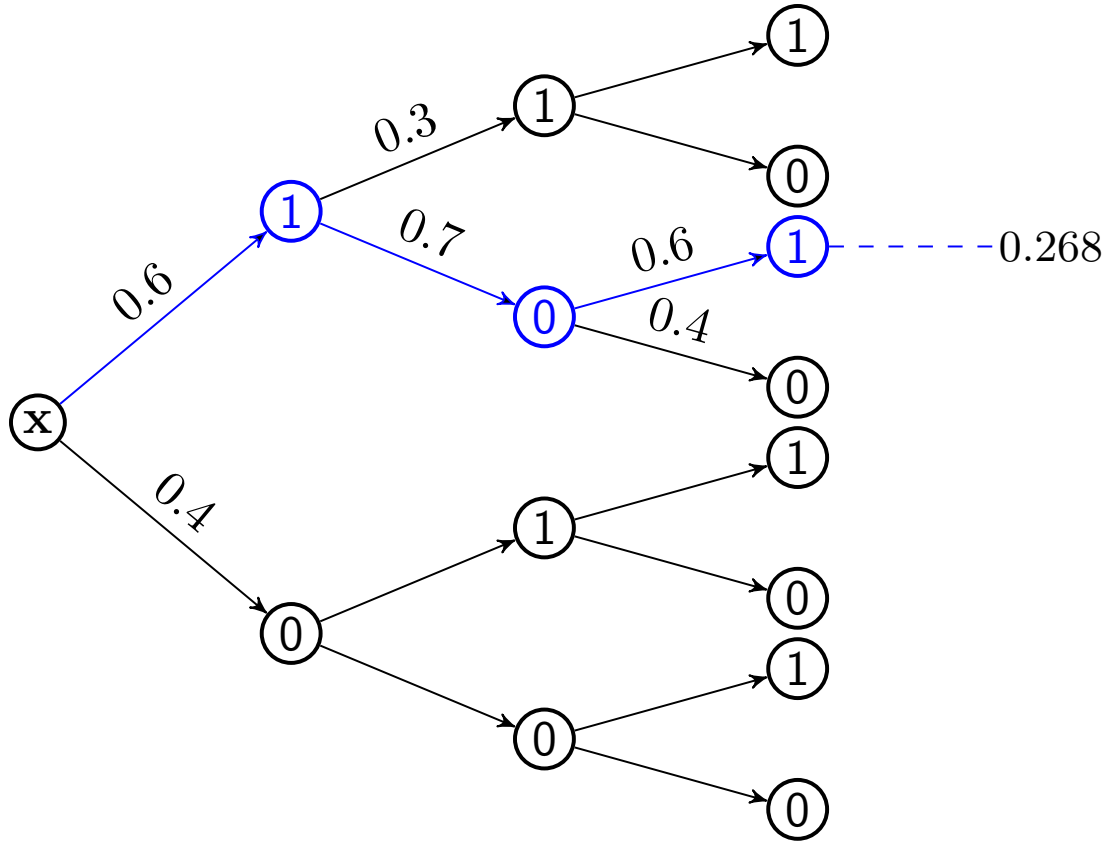
This is not the same as greedy inference.
First problem: Exponential complexity.





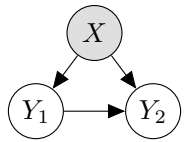
- Exhaustive search $\in \{0,1\}^L$, i.e., 2^L paths,
  Bayes optimal, usually intractable
- Single pass down chain (greedy), 1 path,
  Searches greedily for the joint mode; not optimal.





- Exhaustive search $\in \{0,1\}^L$, i.e., 2^L paths,
  Bayes optimal, usually intractable
- Single pass down chain (greedy), 1 path,
  Searches greedily for the joint mode; not optimal.

First Question: Is Binary Relevance Enough?



Suppose we want to predict and evaluate in particular

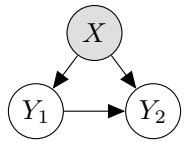
$$\ell(y_2, \widehat{y}_2) = \begin{cases} 1 & y_2 \neq \widehat{y}_2, \\ 0 & y_2 = \widehat{y}_2 \end{cases}$$

$\widehat{y}_1$ does not appear in our query, nor as evidence:

$$P(y_2|\boldsymbol{x}) = \sum_{y_1 \in \{0,1\}} P(y_1|\boldsymbol{x}) P(y_2|\boldsymbol{x}, y_1)$$

i.e., we can model $P(y_2|\boldsymbol{x})$ however we want, directly from $\boldsymbol{x}$.

First Question: Is Binary Relevance Enough?



Suppose we want to predict and evaluate in particular

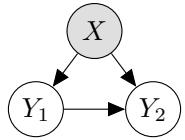
$$\ell(y_2, \widehat{y}_2) = \begin{cases} 1 & y_2 \neq \widehat{y}_2, \\ 0 & y_2 = \widehat{y}_2 \end{cases}$$

$\widehat{y}_1$ does not appear in our query, nor as evidence:

$$P(y_2|\boldsymbol{x}) = \sum_{y_1 \in \{0,1\}} P(y_1|\boldsymbol{x}) P(y_2|\boldsymbol{x}, y_1)$$

i.e., we can model $P(y_2|\boldsymbol{x})$ however we want, directly from $\boldsymbol{x}$.

First Question: Is Binary Relevance Enough?



Suppose we want to predict and evaluate in particular

$$\ell(y_2, \widehat{y}_2) = \begin{cases} 1 & y_2 \neq \widehat{y}_2, \\ 0 & y_2 = \widehat{y}_2 \end{cases}$$

$\widehat{y}_1$ does not appear in our query, nor as evidence:

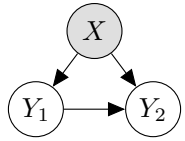
$$P(y_2|\boldsymbol{x}) = \sum_{y_1 \in \{0,1\}} P(y_1|\boldsymbol{x}) P(y_2|\boldsymbol{x}, y_1)$$

i.e., we can model $P(y_2|\boldsymbol{x})$ however we want, directly from $\boldsymbol{x}$.

Now suppose we are minimizing subset 0/1 loss,

$$\ell(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \begin{cases} 1 & \boldsymbol{y} \neq \widehat{\boldsymbol{y}}, \\ 0 & \boldsymbol{y} = \widehat{\boldsymbol{y}} \quad \text{(exactly)} \end{cases}$$

First Question: Is Binary Relevance Enough?



Suppose we want to predict and evaluate in particular

$$\ell(y_2, \widehat{y}_2) = \begin{cases} 1 & y_2 \neq \widehat{y}_2, \\ 0 & y_2 = \widehat{y}_2 \end{cases}$$

$\widehat{y}_1$ does not appear in our query, nor as evidence:

$$P(y_2|\boldsymbol{x}) = \sum_{y_1 \in \{0,1\}} P(y_1|\boldsymbol{x}) P(y_2|\boldsymbol{x}, y_1)$$

i.e., we can model $P(y_2|\boldsymbol{x})$ however we want, directly from $\boldsymbol{x}$.

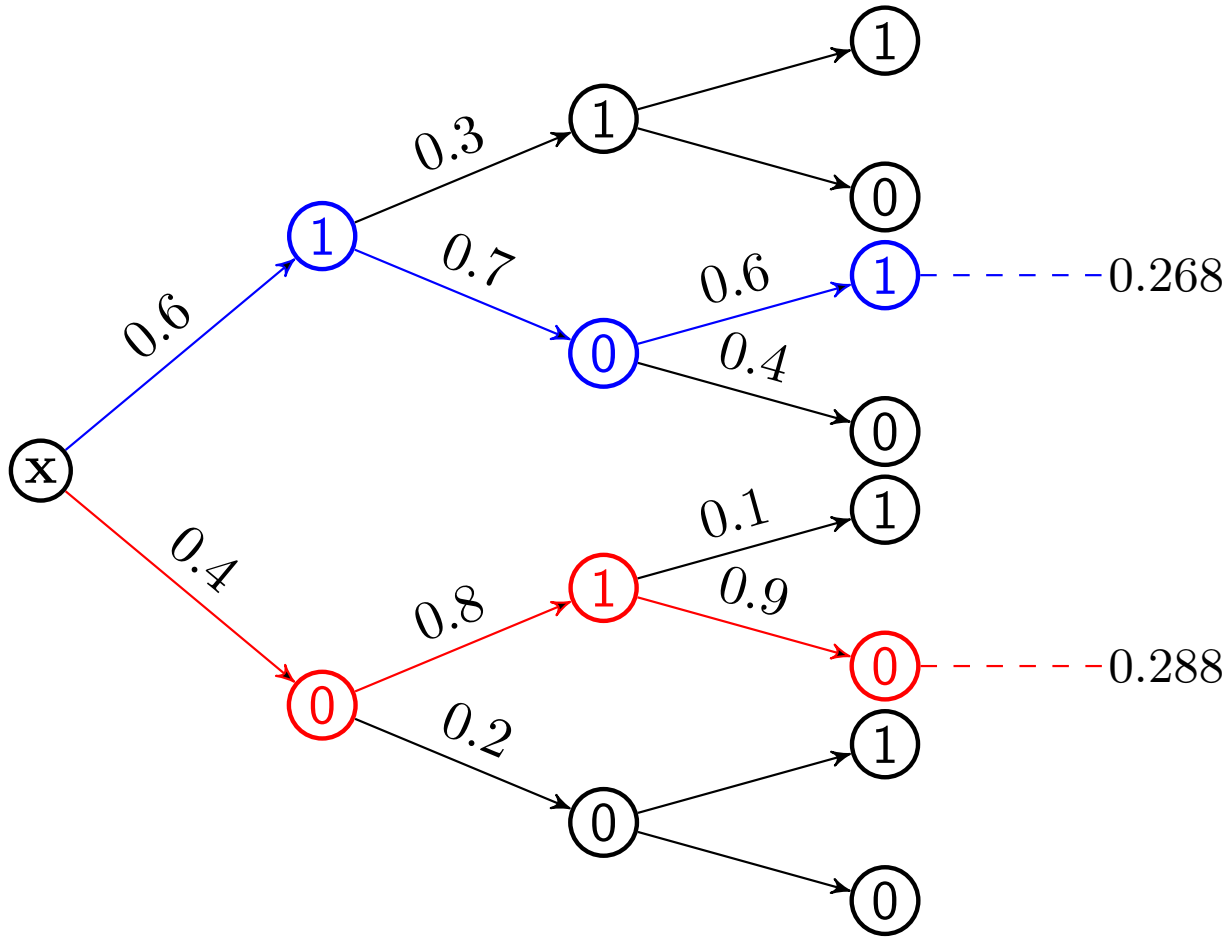
Now suppose we are minimizing subset 0/1 loss,

$$\ell(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \begin{cases} 1 & \boldsymbol{y} \neq \widehat{\boldsymbol{y}}, \\ 0 & \boldsymbol{y} = \widehat{\boldsymbol{y}} \quad \text{(exactly)} \end{cases}$$

Now suppose mean squared error $\mathrm{MSE}(\boldsymbol{y}, \widehat{\boldsymbol{y}})$.

Inference as Search

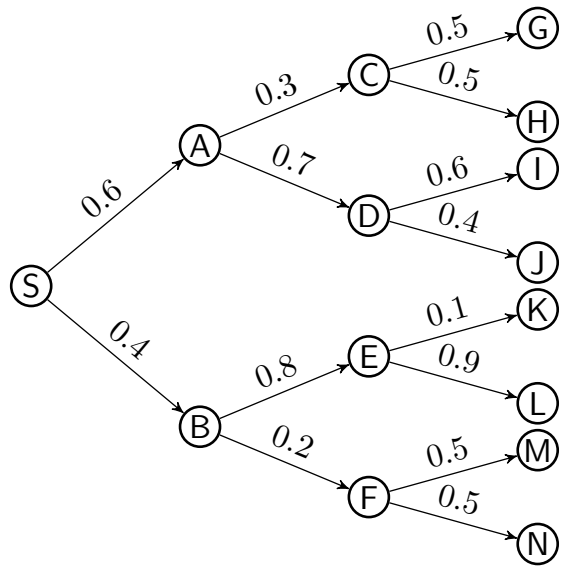
Inference as Tree-Search



- Node $n := y_1, \ldots, y_j \in \{0,1\}^j$; $j \in \{1, \ldots, L\}$ and start state $\emptyset$
- Goal state: Some $\boldsymbol{y} = y_1, \ldots, y_L$
- Edge cost: $P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})$
- Path cost: $g^{-1}(n) = P(y_1, \ldots, y_j | \boldsymbol{x})$. • N.B. $\neq$ num. of steps!

Uniform Cost Search[1] (UCS)

Expanding node n will incur a path-cost $g(n)$.



- A priority queue of nodes and respective path cost $\{(n, g(n))\}$
- Expand lower-cost nodes first (n giving smallest $g(n)$)
- Goal test is applied *when selected for expansion* rather than first encountered (in case a better path there can be found)
- Optimal in general (for costs ≥ 0), i.e., the solution is the best

[1] aka Dijkstra's Algorithm, esp. when expanding *all* nodes

On weighted graph G, from node n_0 to (goal node) n_{goal}:

UniformCostSearch$(G, n_0, n_{\text{goal}})$:

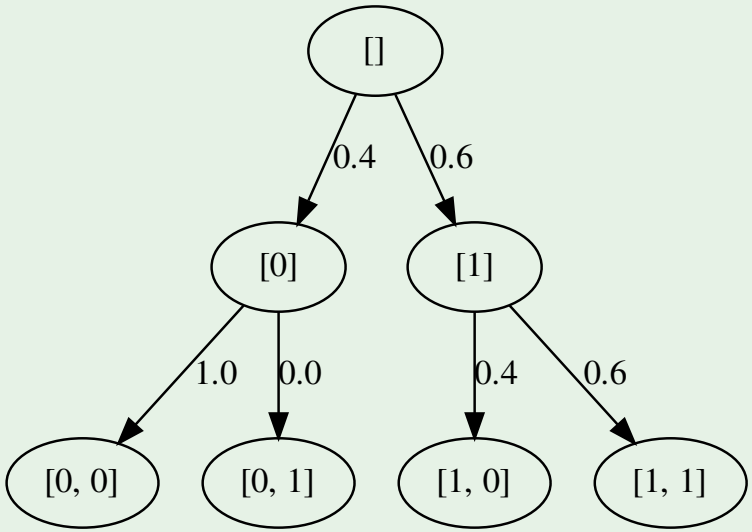
- $\mathcal{Q} = \{(n_0, g(n_0)\}$ ▷ The priority queue
- While $|\mathcal{Q}| > 0$:
  - $n \leftarrow \mathcal{Q}.\text{pop}()$ ▷ Pop[2] n with smallest $g(n)$
  - if $n = n_{\text{goal}}$, stop!
  - for $n' \in \text{Neighbours}(n)$:
    - $g(n') = g(n) + \texttt{cost}(n, n')$
    - $\mathcal{Q} \leftarrow \mathcal{Q} \cup \left(n', g(n')\right)$

(N.B. Usually also want to keep track of the path).

[2] where

$$n^* \leftarrow \mathcal{Q}.\text{pop}() \Leftrightarrow \begin{cases} n^* = \text{argmin}_{n \in \mathcal{Q}}\, g(n), \\ \mathcal{Q} \leftarrow \mathcal{Q} \setminus n^* \end{cases}$$

Uniform Cost Search for Probability-Tree Search



Using queue $\mathcal{Q}$, of tuples $(n, g(n))$,

$\mathcal{Q} = \{([], 0)\}$

$\mathcal{Q} = \{([1], 0.6), ([0], 0.4)\}$ ▷ pop []

$\mathcal{Q} = \{([0], 0.4), ([1, 1], 0.36), ([1, 0], 0.24)\}$ ▷ pop [1]

$\mathcal{Q} = \{([0, 0], 0.4), ([1, 1], 0.36), ([1, 0], 0.24), ([0, 1], 0.0)\}$ ▷ pop [0]

Return: $\boldsymbol{y} = [0, 0]$

Beam Search

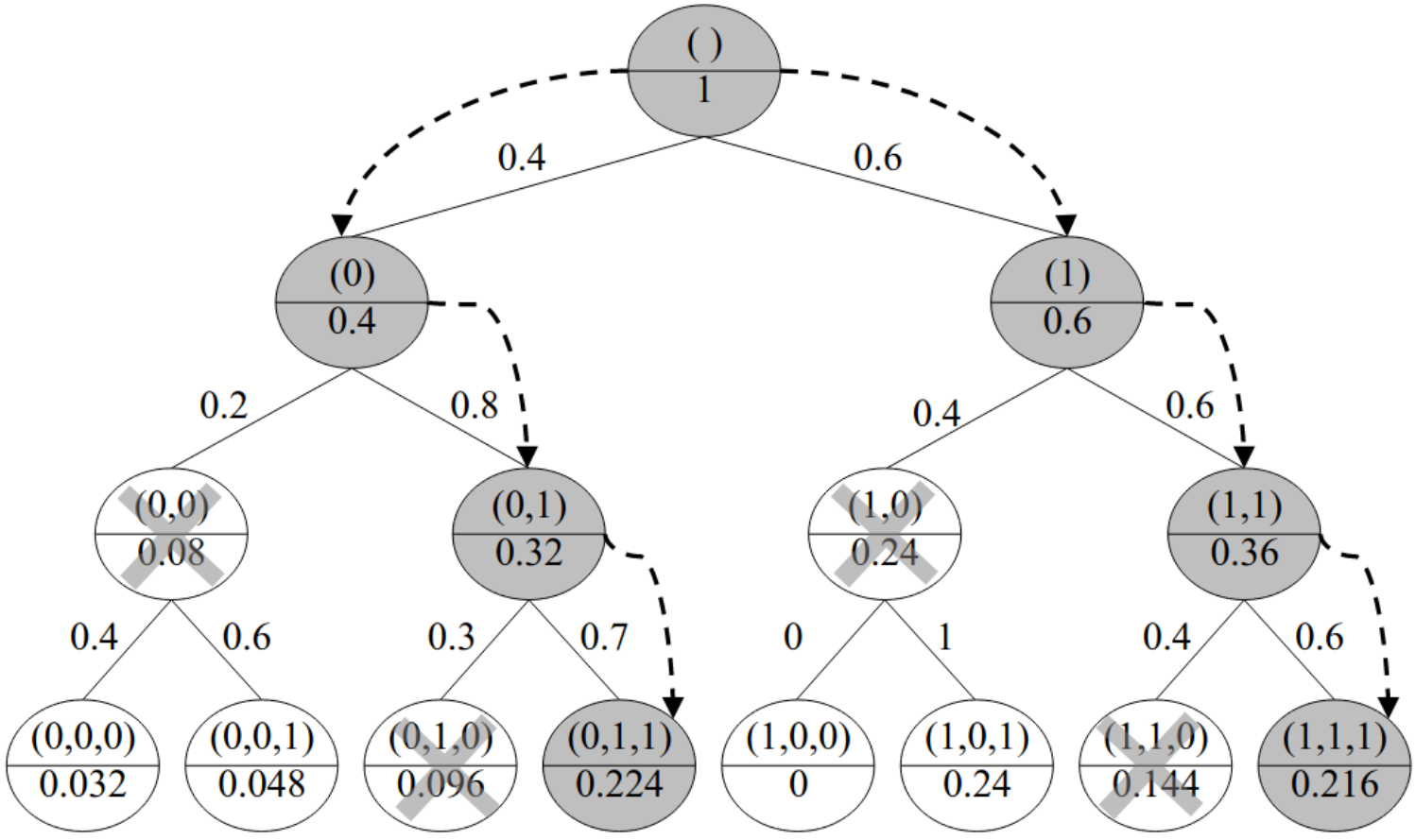
A variant of UCS with reduced memory requirements; only considers the top β nodes (the size of the **beam**) wrt path cost at each level.

- A variant of greedy search.
- Popular in neural networks, especially in language models

Where β is a hyperparameter;

- If $\beta = \infty$, then it is standard UCS

Beam Search, $\beta = 2$



From: Mena et al., *An overview of Inference Methods in Proba- bilistic Classifier Chains for Multi-label classification*

ϵ-Approximate Search

Proceed like UCS (node $[y_1, \ldots, y_j]$ is added to the queue), *but only if*, at depth j,

$$\underbrace{P(y_1, \ldots, y_j | \boldsymbol{x})}_{g(n)} > \epsilon$$

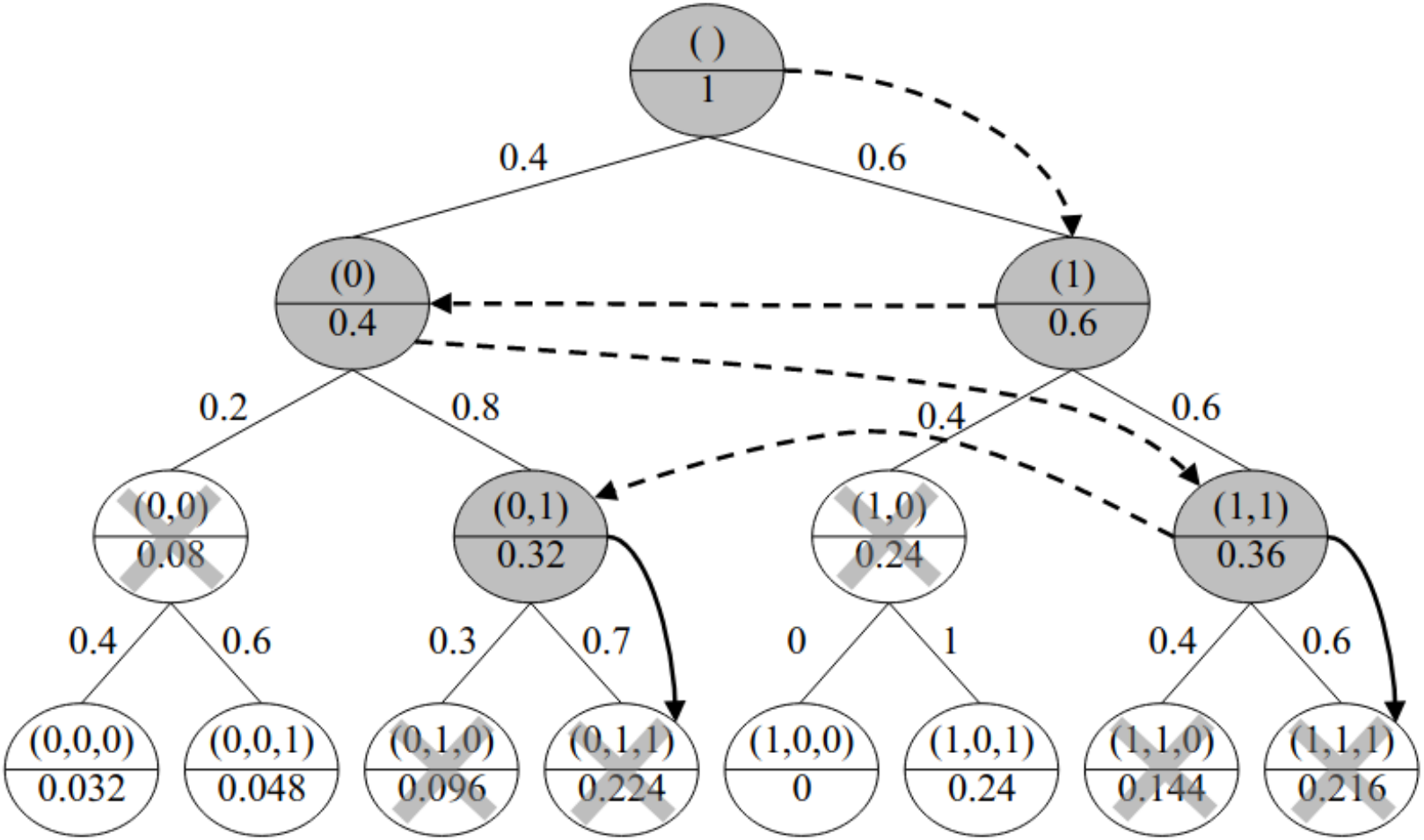
Continue (pop, goal-check, add children), until

- a leaf is found (done!); or
- queue is empty: apply greedy search to goal (from all childless (non-expanded) nodes).

Where ϵ is a hyper-parameter;

- if $\epsilon = 0.0$, then unit costs are ignored; defaults to UCS
- if $\epsilon = 0.5$, defaults to greedy search.

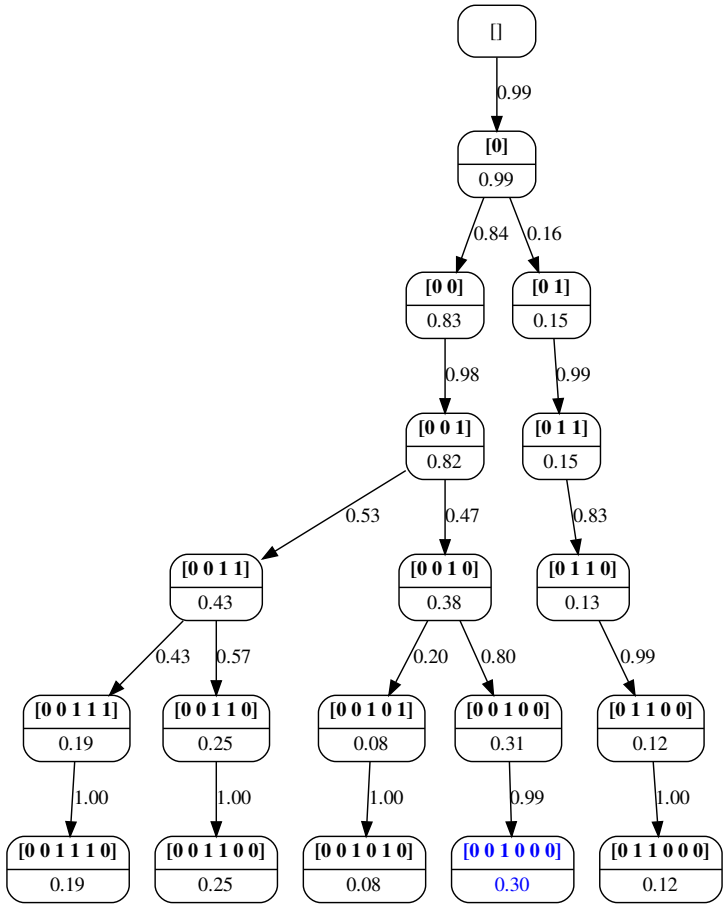
ϵ-Approximate Search, $\epsilon = 0.25$.



From: Mena et al., *An overview of Inference Methods in Proba- bilistic Classifier Chains for Multi-label classification*

Monte Carlo Search

1. Take M samples/paths from the search tree from root to leaf i.e., open branch $y_j^{(m)}$ with prob. $P(y_j^{(m)}|\boldsymbol{x}, y_1^{(m)}, \ldots, y_{j-1}^{(m)})$
2. Return the m^*-th path; where $m^* = \text{argmax}_m P(\boldsymbol{y}^{(m)}|\boldsymbol{x})$.



On the Emotions/Music dataset: $M = 20$, only explored branches shown

Recall: We were optimizing 0/1 loss.

- Chance of finding Bayes-optimal solution depends on $\beta/\epsilon/M$
- Other search methods are applicable[3], e.g., A* search, ...

[3] Survey:
Mena et al., *An overview of inference methods in probabilistic classifier chains for multilabel classification*

Recall: We were optimizing 0/1 loss.

- Chance of finding Bayes-optimal solution depends on $\beta/\epsilon/M$
- Other search methods are applicable[3], e.g., A* search, ...

The End ?

But – Empirical observation: Classifier chains also outperforms baseline methods under Hamming loss (where labels evaluated independently); a decomposable metric – risk minimization tells us that clasifier chains provides no advantage).

[3] Survey:
Mena et al., *An overview of inference methods in probabilistic classifier chains for multilabel classification*

Benefits of Structure

Why Model Labels Together

Other reasons to model labels together (other than 'because of label dependence/to minimise 0/1-loss'):

- Connectivity $\Rightarrow$ greater efficiency (sometimes)
- Connectivity $\Rightarrow$ better interpretation
- Connectivity $\Rightarrow$ regularization
- Connectivity $\Rightarrow$ predictive power (deep learning!)
- Connectivity $\Rightarrow$ suited to more loss metrics

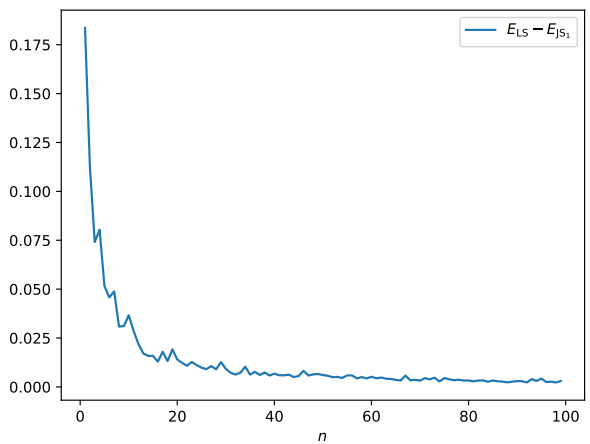
Reasons **not** to model targets together: greater efficiency (sometimes), simplicity, parallelizability, ...

Reason #1: Regularization

Joint-target regularization (modeling labels together) is beneficial even if targets are intrinsically independent.

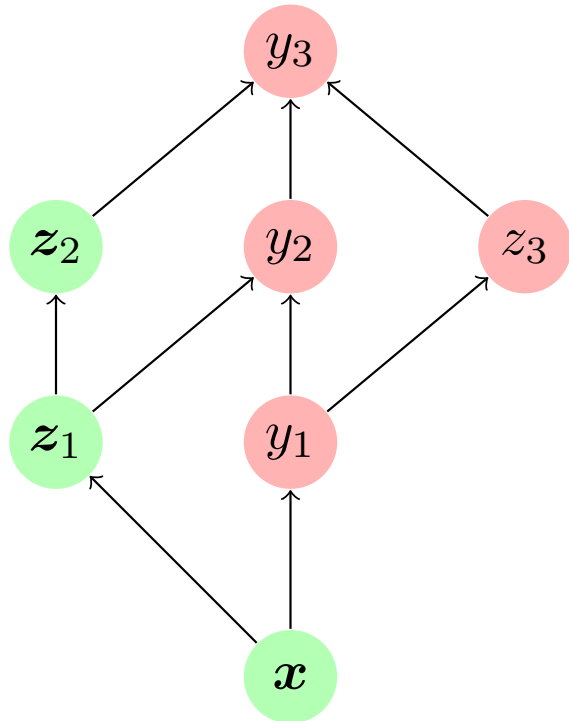
James-Stein Estimator (m targets, n samples)

$$\begin{aligned}\widehat{\boldsymbol{y}}_{\mathrm{JS}_n} &= \frac{1-(m-2)\frac{\hat{\sigma}^2}{n}}{\|\bar{\boldsymbol{y}}\|^2}\cdot\bar{\boldsymbol{y}} \\ &= \lambda(\{\boldsymbol{y}^{(1)},\ldots,\boldsymbol{y}^{(n)}\})\cdot\widehat{\boldsymbol{y}}_{\mathrm{MLE}}\end{aligned}$$



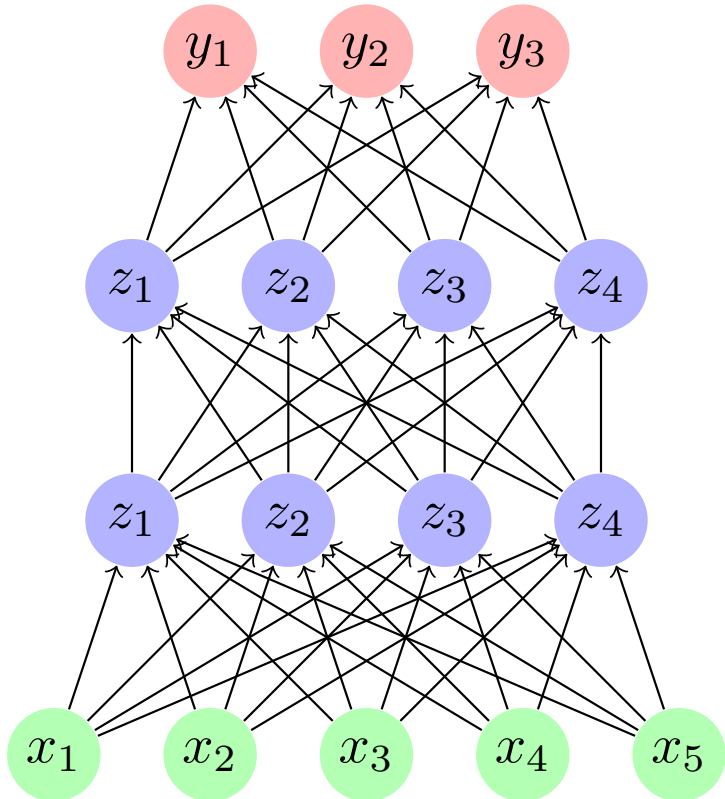
But the advantages quickly fade as $n \gg 1$.

Reason #2: Predictive Power/Architecture

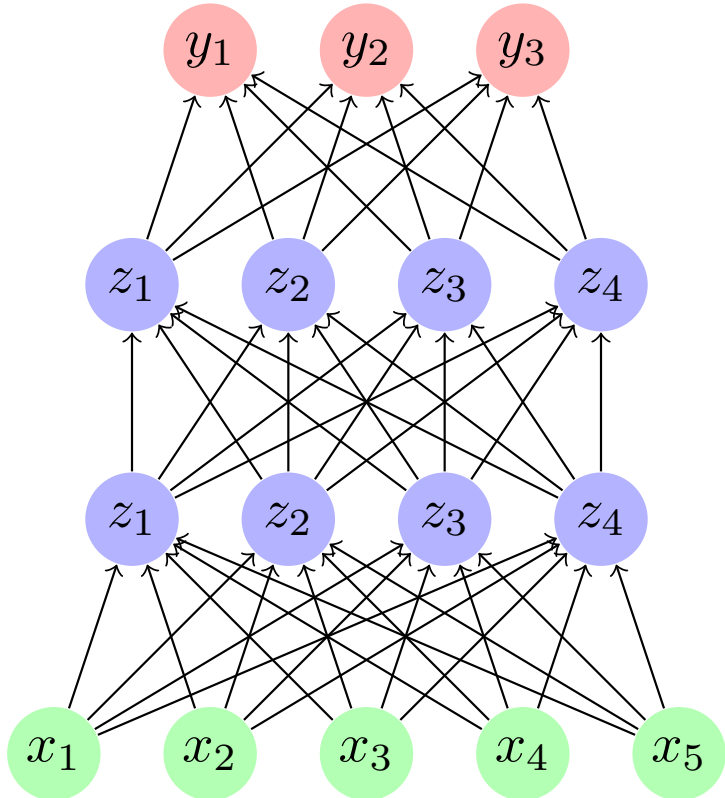


with delay nodes $z = f(x) = x$

- Forward propagation = greedy inference
- It's deep in the label space!
- labels = feature space/hidden nodes 'for free'
- More parameters $\Rightarrow$ more powerful model



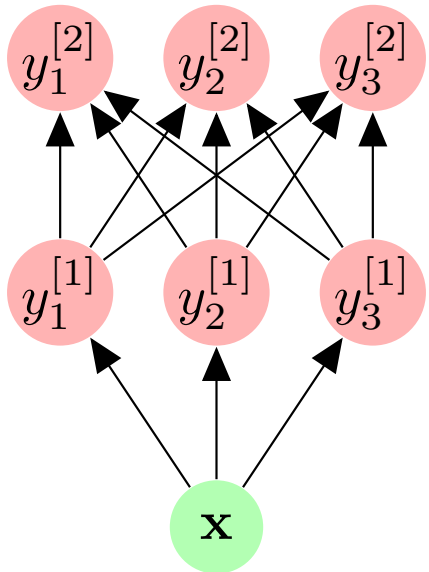
(This also works, but we have to learn $\boldsymbol{z}^{[l]}$ from scratch.)



(This also works, but we have to learn $\boldsymbol{z}^{[l]}$ from scratch.)
There are 'deep' chaining-type architectures; consider, e.g., ResNet, skip layers, etc.

Correcting Errors/Bias with Stacking

In multi-label stacking, the predictions are stacked as features.



$y_j^{[l]} = j$-th label, l-th layer

Like classifier chains at inference time: $\widehat{y}_j^{[1]}$ is used to predict $\widehat{y}_k^{[2]}$, but important difference in training: predictions $\widehat{y}_j^{(i)}$ are used!

- Using dataset labels: model conditional dependencies, better under 0/1 loss
- Using predicted labels: extend feature space, improve under Hamming loss

Comparison

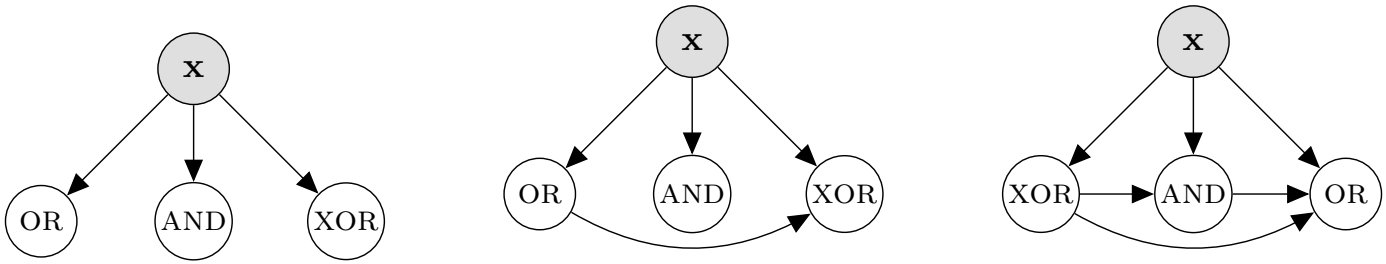
	X_1	X_2	X_3	Y_2
Basis expansion	x	ϕ_1	ϕ_2	y_2
Classifier chain	x	y_1		y_2
Stacking	x	$\widehat{y}_1$	$\widehat{y}_2$	y_2
Neural network	x			y_2

X_1	X_2	X_3	Y_2
$\tilde{x}$	ϕ_1	ϕ_2	$\widehat{y}_2$
$\tilde{x}$	$\widehat{y}_1$		$\widehat{y}_2$
$\tilde{x}$	$\widehat{y}_1^{[1]}$	$\widehat{y}_2^{[1]}$	$\widehat{y}_2^{[2]}$
$\tilde{x}$	$\hat{z}_1$	$\hat{z}_2$	$\widehat{y}_2$

Training (left; $x \equiv x^{(i)}$, $y_j \equiv y_j^{(i)}$) vs Testing (right; $\widehat{y}_2|\tilde{x}$)

Which Structure?

OK, yes, structure works – but which structure?



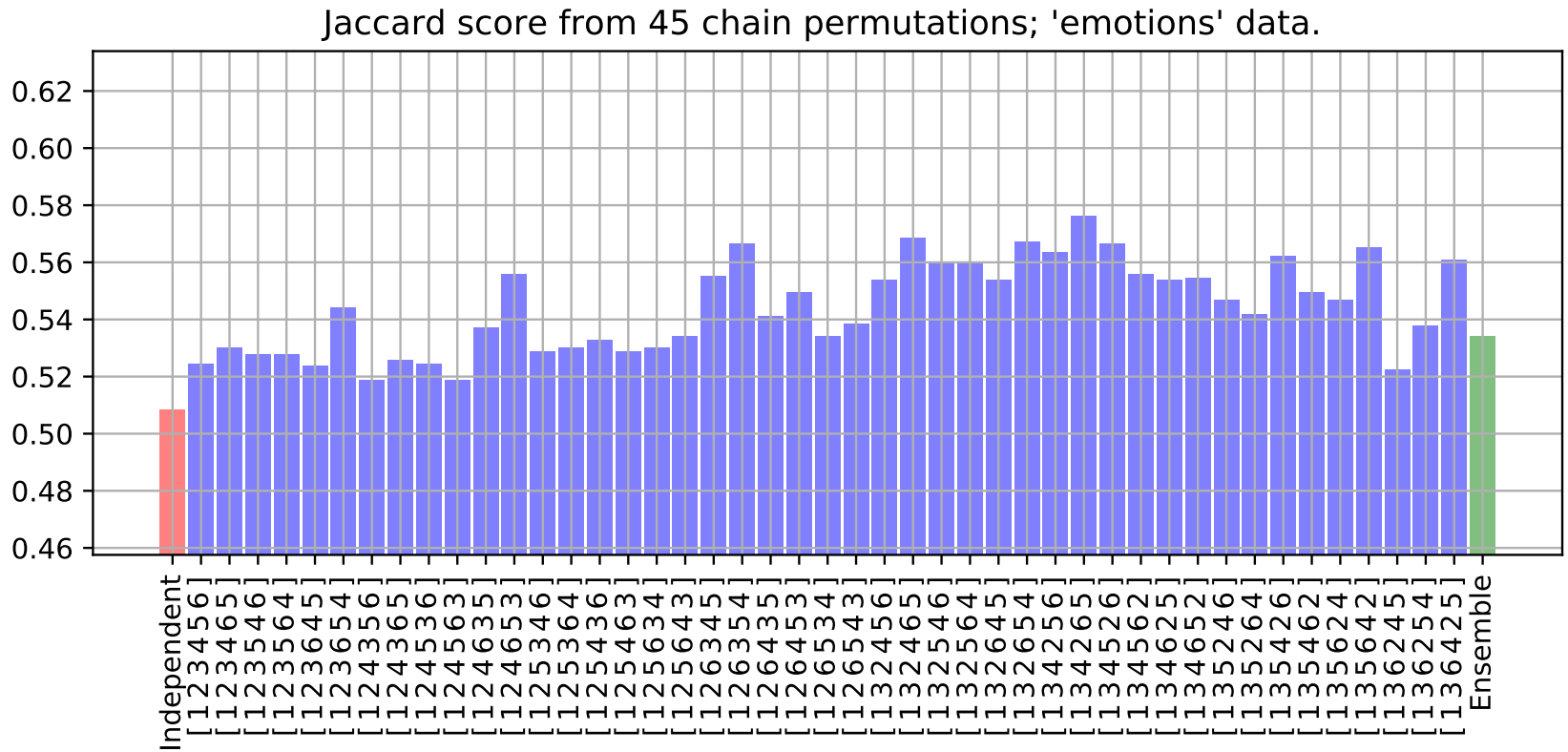
$\boldsymbol{x} \in \{0, 1\}^2$, and labels are logical operations

Consider:

- Choice of base classifier
- Choice of inference
- What is your loss metric?
- Will we find ground truth? Do we need to?

Structure from Data / Learning Structure as a Search

Motivation: Structure Can Make a Difference

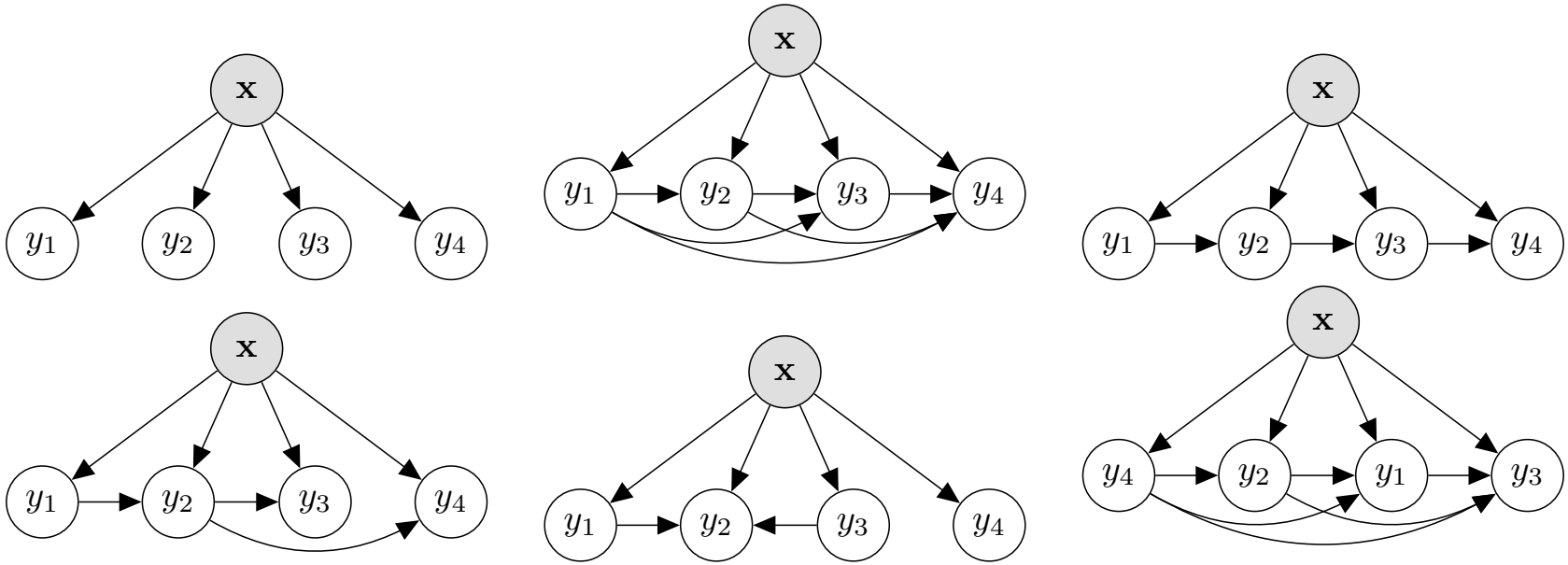


Highlighting some example cases – What is going on?

m	Chain Order	Jaccard	A1	A2	A3	A4	A5	A6
5	[1 2 3 6 4 5]	0.52	0.81	0.70	0.73	0.93	0.82	0.74
6	[1 2 3 6 5 4]	0.54	0.81	0.70	0.73	0.93	0.86	0.74
17	[1 2 5 4 6 3]	0.53	0.81	0.70	0.73	0.94	0.81	0.73
18	[1 2 5 6 3 4]	0.53	0.81	0.70	0.71	0.91	0.81	0.77
20	[1 2 6 3 5 4]	0.57	0.81	0.70	0.75	0.95	0.84	0.77
42	[1 3 5 6 4 2]	0.57	0.81	0.72	0.74	0.94	0.83	0.76

Jaccard score and per-label accuracy (Aj) for all labels $j = 1, \ldots, 6$

Options for Structure



1. Random structure (often in an ensemble).
2. Use an existing hierarchy (expert knowledge)
3. Use a full/complete structure
4. Impose a fixed structure allowing tractable inference
5. Learning a structure via procedure/search based on a score of
   - Heuristics based on label dependence,
   - Individual accuracy
   - Pairwise accuracy
   - Structure accuracy
   - ...

Learning Structure from Data

Assume training data

$$\mathcal{D} \sim P_{G^*}$$

(is generated from ground-truth graph $G^* \in \mathcal{G}$).

Unfortunately:

- Intractable number of graphs $|\mathcal{G}|$ (super-exponential)
- And we cannot identify G^* anyway
  - Not enough data; instability (different G for same $\mathcal{D}$)
  - Different G for same P (expression of joint dist.)

Fortunately:

- We can recover some equivalent G / equivalence class of G^*
- We often don't need G^* (just some G for good accuracy)
- Many approaches are available to help us search for it

Search for Structure

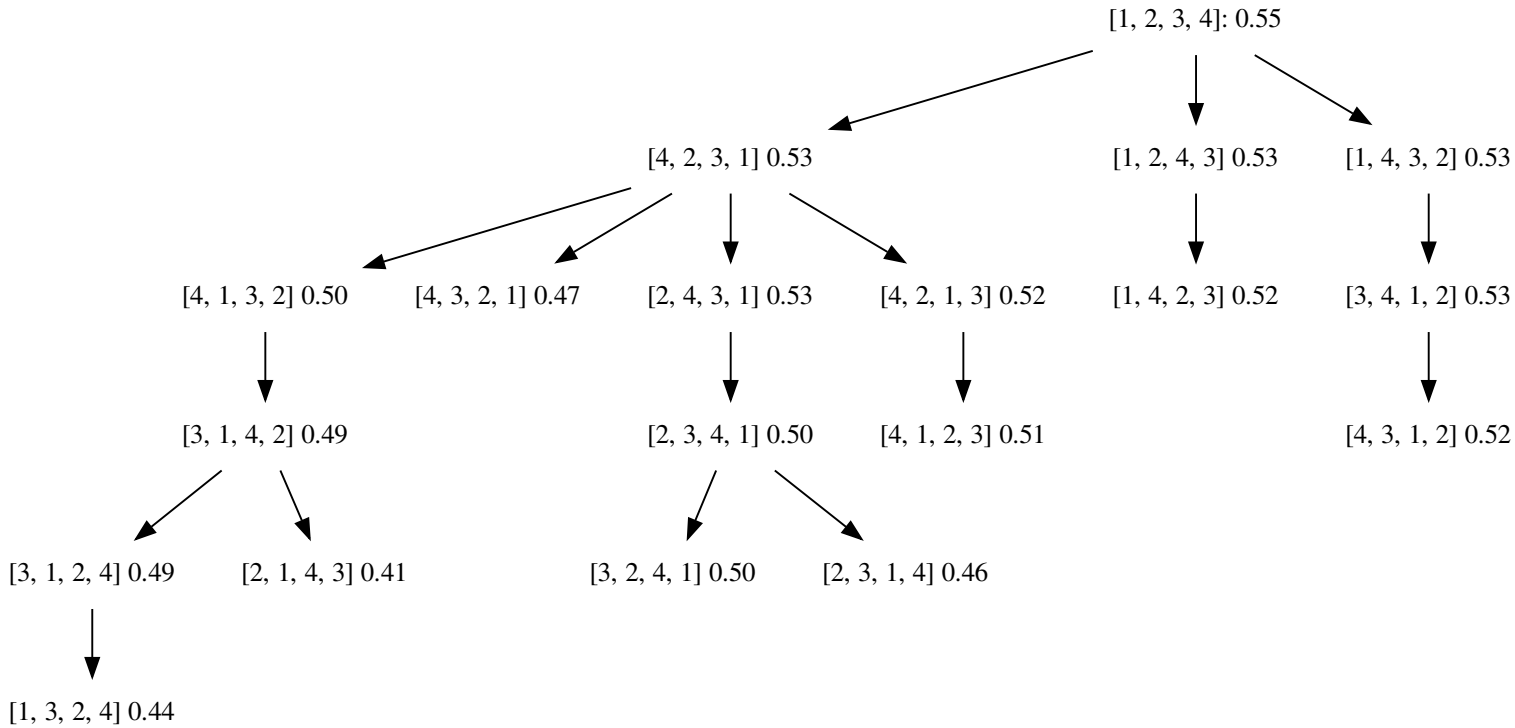
- Search space: $\mathcal{G}$ (all possible graphs/structures)
- Nodes n (let n identify a graph $G_n \in \mathcal{G}$)
- Node cost $g(n) \approx$ expected loss incurred by our model h_{G_n}

Our search problem:

$$\hat{G} = \underset{G_n \in \mathcal{G}}{\operatorname{argmin}} \underbrace{\mathbb{E}[\ell(h_{G_n}(\boldsymbol{X}), \boldsymbol{Y})]}_{\approx g(n)}$$

- We can apply the same strategies used for inference (UCS, Beam-Search, Monte Carlo search, ...) and others.
- Expensive (must approximate $\mathbb{E}$, implies building models)
- But there are tricks, e.g., search only for order, freeze the graph from root to leaf during the search.

Remark/reminder: consideration of our cost function $g(n)$, ℓ is important!


[1, 2, 3, 4]: 0.55
[4, 2, 3, 1] 0.53
[1, 2, 4, 3] 0.53
[1, 4, 3, 2] 0.53
[4, 1, 3, 2] 0.50
[4, 3, 2, 1] 0.47
[2, 4, 3, 1] 0.53
[4, 2, 1, 3] 0.52
[1, 4, 2, 3] 0.52
[3, 4, 1, 2] 0.53
[3, 1, 4, 2] 0.49
[2, 3, 4, 1] 0.50
[4, 1, 2, 3] 0.51
[4, 3, 1, 2] 0.52
[3, 1, 2, 4] 0.49
[2, 1, 4, 3] 0.41
[3, 2, 4, 1] 0.50
[2, 3, 1, 4] 0.46
[1, 3, 2, 4] 0.44

Dynamic Structures

So far:

1. Find $\hat{G} \in \mathcal{G}$, train $h_{\hat{G}}$
2. At inference time,

$$\hat{\boldsymbol{y}} = h_{\hat{G}}(\tilde{\boldsymbol{x}}) = \underset{\boldsymbol{y} \in \mathcal{Y}}{\operatorname{argmax}} P_{\hat{G}}(\boldsymbol{y} | \tilde{\boldsymbol{x}})$$

but is $\hat{G}$ best for test $\tilde{\boldsymbol{x}}$ in particular?

- Structure may be conditionally dependent on test instance!
- We can include structure in inference;

$$\hat{\boldsymbol{y}} = h(\tilde{\boldsymbol{x}}) = \underset{\boldsymbol{y} \in \mathcal{Y}, G \in \mathcal{G}}{\operatorname{argmax}} P_G(\boldsymbol{y} | \boldsymbol{x})$$

approximations are necessary; many trade-offs involved!

Ideal setting: an agent looks at the data, provides the appropriate structure and parameters (architecture).

Neural Architecture Search

- Structure search is not specific to Bayesian networks
- Neural networks face the same problem: how to know what structure (architecture) and [hyper]-parameters to use, for a given problem/data set
- Bayesian networks can be neural networks (e.g., classifier chains); (Generally: Bayesian networks marginalize out $\boldsymbol{z}$, Neural networks propagate through $\boldsymbol{z}$)

Some solutions/related concepts to be aware of:

- Use a recurrent structure (e.g., RNNs, LSTMs; output labels as a sequence)
- Transformers and attention mechanisms
- Hyperparameter optimization (structure as a hyperparameter)
- Meta learning: learning to learn (including structure)
- Automated ML (AutoML): structure choice is automated
- Neural Architecture Search: a policy to decide how build and traverse structure; reinforcement learning.

Summary

Summary

Machine learning/AI applications are increasingly multi- and structured-output. Models often require structure.

How to address this issue? Take a principled approach! –

- Do you need structure? (consider your loss metric)
- What purpose will it serve? (architecture for predictive power? space efficiency? regularization? interpretation?)
- Is one structure from training sufficient? Or do you need several structures, dynamically?
- How will you traverse your structure (what type of inference)?

Probabilistic vs [deep] neural-network architectures:

- Much in common, can be equivalent/part of the same model
- Different perceptions of "depth": layers vs iterations

Everything a search: Learning (find $\hat{G}, \hat{\theta}$) and Inference (find $\widehat{\boldsymbol{y}}$):

$$\widehat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \{0,1\}^L, G_\theta \in \mathcal{S}}{\operatorname{argmax}} P_{G_\theta}(\boldsymbol{y}|\boldsymbol{x})$$

Structure benefits agents and agents benefit structure [search]!

Multi-Target and Structured-Output Learning

INF581 Advanced Machine Learning and Autonomous Agents

Jesse Read